INT. J. PROD. RES., 2000, VOL. 38, NO. 10, 2173–2184

Scheduling a batch processing machine with non-identical job sizes

MERAL AZIZOGLU† and SCOTT WEBSTER‡*

The problem of scheduling batch processors is important in some industries and, at a more fundamental level, captures an element of complexity common to many practical scheduling problems. We describe a branch and bound procedure applicable to a batch processor model with arbitrary job processing times, job weights and job sizes. The scheduling objective is to minimize total weighted completion time. We find that the procedure returns optimal solutions to problems of up to ~ 25 jobs in reasonable CPU time, and can be adapted for use as a heuristic for larger problems.

1. Introduction

This paper addresses the following model of a batch processor scheduling problem. Multiple jobs are processed simultaneously in batches, and the processing time of a batch is equal to the longest processing time of the jobs in the batch. The processor space requirements may vary by job and the scheduling objective is to minimize total weighted completion time. This type of problem arises in a number of different settings including diffusion and burn-in operations in semiconductor fabrication, heat treatment operations in metalworking, and firing operations in ceramics.

The model is important for two main reasons. First, the economic impact of one of the motivating applications of the model is significant. The burn-in operation is where semiconductors are exposed to high temperatures to weed out chips susceptible to premature failure (Uzsoy *et al.* 1992). This operation is generally one of the keys to effective scheduling of semiconductor fabrication plants, scheduling is a significant determinant of efficiency and responsiveness in this industry, and the industry plays a meaningful role in the overall economy. Effective burn-in operation scheduling is a key because it is frequently a bottleneck due to long processing times relative to other operations (e.g. days versus hours) and because it occurs at the end of the manufacturing process and thus has a strong influence on ship dates (e.g. there is little room to compensate for delays in burn-in by expediting subsequent operations). Relative to other manufacturing environments, effective use of resources in semiconductor fabrication tends to be more critical due to the very high cost of equipment. Finally, the role this industry plays in the overall economy is considerable because the computer revolution is being driven by the ability to economically develop and manufacture semiconductors, and semiconductors are being used in an increasing variety of everyday products. A recent issue of *Business Week* (8/25/97, p.

Revision received December 1999.

† Middle East Technical University, Department of Industrial Engineering, Ankara 06531, Turkey.

‡ Syracuse University, School of Management, Syracuse, NY 13244-2130, USA.

* To whom correspondence should be addressed. e-mail: stwebste@syr.edu

International Journal of Production Research ISSN 0020–7543 print/ISSN 1366–588X online
http://www.tandf.co.uk/journals

66), for example, observes that during the past 4 years 'the mere production of computers and semiconductors accounted for 45% of US industrial growth.'

Second, the batch processor model embodies a basic source of tension present in many practical scheduling problems—efficiency versus timely completion of individual jobs. On one hand, partitioning jobs into batches so that all jobs in a batch have the same processing time and use all available space in the processor maximizes efficiency. On the other hand, higher priority jobs should be processed before lower priority jobs. Insights into how to deal with various manifestations of this conflict may eventually lead to practical solution methods for more complex and more widely applicable models of scheduling problems in industry.

Early work on batch processor models can be traced to Ikura and Gimple (1986) who propose an optimal algorithm for a batch processor model with identical job processing times, identical job sizes, dynamic job arrivals and an objective of minimizing makespan. Several researchers have studied batch processor models with the weighted completion time criterion. Chandru *et al.* (1993a, 1993b), DuPont and Ghazvini (1997), Hochbaum and Landy (1997a, 1997b), and Brucker *et al.* (1998) consider the special case where job sizes and weights are identical. Brucker *et al.* (1998) and Uzsoy and Yang (1997) propose solution methods for the more general model where job weights are allowed to be arbitrary, but job sizes are assumed to be identical. Uzsoy (1994) proposes an optimization procedure and several fast heuristics for a model where job sizes are allowed to be arbitrary, but job weights are assumed to be identical. This problem is known to be NP-hard (Uzsoy 1994), whereas the complexity of the problem with identical job sizes and either identical or arbitrary job weights remains open.

Researchers have studied batch processor models under a variety of objective functions where jobs are partitioned into families, jobs from a family have the same processing time, and batches are limited to jobs from a single family (i.e. incompatible job families; see Uzsoy 1995, Mehta and Uzsoy 1998). Other variations on the models described in the previous paragraph include consideration of various due date-based objectives (Brucker *et al.* 1998).

The model we are considering generalizes the model in Uzsoy (1994) to the case of arbitrary job weights, and generalizes the model in Uzsoy and Yang (1997) to the case of arbitrary job sizes. This generalization adds a level of complexity by bringing the tension between efficiency and timely completion more squarely into conflict. The intuition into why this is so stems from two observations: (i) batching efficiency is maximized if jobs in each batch all have the same processing time and consume all the available space in the processor; and (ii) such a batching is also perfect with respect to timely completion of individual jobs if job weights are identical (i.e. relative job urgency is proportional to processing time so that batches will be sequenced in shortest processing time order). Consequently, the efficient 'packing' of jobs into batches is complicated not only by varying job processing times and sizes, but also because jobs of similar processing time may not be of similar urgency. To our knowledge, algorithms for the model we are considering have not appeared in the literature.

The main contribution of this paper is the design of a branch and bound (B&B) algorithm. The problem we are considering is NP-hard (follows from the NP-hardness of the unit weight problem, see Uzsoy 1994), which suggests that any optimization procedure will run into computational difficulties at some point as the number of jobs increases. There is, however, the practical question concerning the size of

problems that are solvable in a reasonable amount of time. Our computational tests suggest that the answer to this question for our algorithm is ~ 25 jobs. Furthermore, because our B&B algorithm employs a depth-first strategy (i.e. very quickly generates feasible schedules), it may be employed as a heuristic for larger problems by terminating with the best solution after a set amount of CPU time.

An important aspect of our procedure concerns a set of dominance properties that are identified in section 2. We find that the use of these properties to fathom nodes in the B&B tree leads to significant speed up of the algorithm. The specifics on how these properties are used, as well as other details of the algorithm design are specified in section 3. Section 4 reports results from computational tests and section 5 concludes the paper.

2. Model, properties and bounds

2.1. *Parameters*

- B processor capacity,
- n number of jobs,
- a_j size of job j,
- p_j processing time of job j, indexed such that $p_1 \leq p_2 \leq \cdots \leq p_n$,
- w_j weight of job j where $w_j > 0$.

2.2. *Variables and functions*

- B_i set of job indices assigned to ith batch in a schedule,
- S schedule of t batches (i.e. batch start time = completion time of previous batch),
  $= \{B_1, B_2, \ldots, B_t\}$,
- $a(B_i)$ size of batch B_i,
  $= \sum_{j \in B_i} a_j$,
- $p(B_i)$ processing time of batch B_i,
  $= \max_{j \in B_i}\{p_j\}$,
- $w(B_i)$ weight of batch B_i,
  $= \sum_{j \in B_i} w_j$,
- $b(j)$ function that returns the index of the batch where job j is scheduled,
  $= \{i | j \in B_i\}$,
- Ψ set of feasible schedules,
  $= \{S | \bigcup_{i=1}^{t} B_i = \{1,2,\ldots,n\}, B_i \cap B_k = \varnothing \, \forall i \neq k, a(B_i) \leq B \, \forall i\}$,
- $C_j(S)$ completion time of job j according to schedule S,
  $= \sum_{i \leq b(j)} p(B_i)$.

The problem is to find a schedule $S^* \in \Psi$ satisfying

$$\sum_j w_j C_j(S^*) = \mathrm{Min}_{S \in \Psi}\{\sum_j w_j C_j(S)\}.$$

2.3. *Properties*

Property 1 (Uzsoy and Yang 1997)**:** There exists an optimal schedule in which batches are ordered from smallest to largest $p(B_i)/w(B_i)$.

Property 2: If $B - a(B_{b(j)}) \geq a_k$ and $b(k) > b(j)$, then $p_k > p(B_{b(j)})$ in any optimal schedule.

Proof: Suppose we have an optimal schedule with $B - a(B_{b(j)}) \geq a_k$, $b(k) > b(j)$, and $p_k \leq p(B_{b(j)})$. Then job k could be removed from batch $b(k)$ and inserted into batch $b(j)$ without increasing the completion time of any other job and lowering the completion time of job k, thus lowering the objective, which is a contradiction. □

Property 3: If $p_k \leq p_j \leq p(B_{b(k)})$, $w_k > w_j$, $B - a(B_{b(j)}) + a_j \geq a_k$, and $B - a(B_{b(k)}) + a_k \geq a_j$ then $b(j) \geq b(k)$ in any optimal schedule.

Proof: Suppose S^* is an optimal schedule with $p_k \leq p_j \leq p(B_{b(k)})$, $w_k > w_j$, $B - a(B_{b(j)}) + a_j \geq a_k$, $B - a(B_{b(k)}) + a_k \geq a_j$, and $b(j) < b(k)$. Let S' be the schedule that results from interchanging jobs j and k. S' is feasible because $B - a(B_{b(j)}) + a_j \geq a_k$, $B - a(B_{b(k)}) + a_k \geq a_j$. From $pk(pj(p(Bb(k))$ and $b(j) < b(k)$, it follows that $C_k(S' \leq C_j(S^*)$, $C_j(S') \leq C_k(S^*)$, and $C_t(S') \leq C_t(S^*)$ for all $t \neq j$. This result combined with $w_k > w_j$ implies $\sum_i w_i C_i(S') < \sum_i w_i C_i(S^*)$, which is a contradiction. □

Let F_m^* denote the optimal weighted completion time for a different problem where jobs with processing times $p_1, p_2, \ldots, p_n$ and weights $w_1, w_2, \ldots, w_n$ are to be scheduled on m identical parallel processors each capable of processing a single job at a time. The following property concerns a relationship between F_m^* and $\sum_j w_j C_j(S^*)$.

Property 4: For optimal schedule $S^* = \{B_1, B_2, \ldots, B_t\}$, if $|B_i| \leq m$ for all i, then $F_m^* \leq \sum_j w_j C_j(S^*)$.

Proof: The result can be verified by observing that no job in batch i can begin processing until all jobs in batch $i-1$ have completed, whereas this constraint is relaxed in the parallel processor weighted completion time problem. □

2.4. *Upper bound*

Uzsoy (1994) proposes and evaluates five heuristics for the model with identical job weights. The extended greedy ratio heuristic was found to be most effective in computational tests, and we extend elements of this heuristic to the case of arbitrary job weights. The heuristic successively adds jobs to a batch, creating a new batch when no unassigned job can fit. Among jobs that can fit in a batch, the job that increases the batch processing time to weight ratio by the smallest amount is selected. The procedure is stated below.

Step 0. $r = 0$, $J = \{1, 2, \ldots, n\}$.
Step 1. $r = r + 1$, $B_r = \varnothing$.
Step 2. Let σ be the set of unscheduled jobs that fit in batch r, i.e. $\sigma = \{i \in J | a(B_r) + a_i \leq B\}$. If σ is empty, then go to step 1; otherwise let $k = \text{argmin}_{j \in \sigma}\{\max\{p_j, p(B_r)\}/[w_j + w(B_r)]\}$.
Step 3. $B_r = B_r \cup \{k\}$, $J = J/\{k\}$. If J is not empty, then go to step 2.
Step 4. Sequence the batches in order of smallest to largest $p(B_i)/w(B_i)$.

2.5. *Lower bounds*

Let $\iota_\cdot(k)$ be a function that returns the index of the job with the kth smallest value of parameter $\cdot$. For example, $a_{\iota_a(1)} \leq a_{\iota_a(2)} \leq \cdots \leq a_{\iota_a(n)}$. Let m be defined such that $\sum_{k=1}^{m} a_{\iota_a(k)} \leq B < \sum_{k=1}^{m+1} a_{\iota_a(k)}$. Then $|B_i| \leq m$ for all i in any feasible schedule and, from Property 4, F_m^* is a lower bound on $\sum_j w_j C_j(S^*)$. Determining F_m^* is NP-

hard so we will make use of the following efficient method for computing a lower bound on F_m^* (Azizoglu and Kirca 1999).

$$LB_{AK} = \sum_{j=1}^{n} \left[w_{\iota_w(n-j+1)} \left(C_j - p_j \right) + w_j p_j \right] \quad \text{where} \quad C_j = \begin{cases} p_j, & j \le m \\ C_{j-m} + p_j, & j > m. \end{cases}$$

Schwiegelshohn *et al.* (1998) propose a lower bound for a shelf scheduling problem that is also a valid lower bound for the batch processor scheduling problem (Webster 2000), i.e.

$$LB_S = F_1^*/B + F_n^*/2 - R/2,$$

where F_1^* = optimal total weighted completion time of a single processor problem with job processing times $a_j p_j$ and weights w_j, $F_n^* = \sum_j w_j p_j$, and $R = \sum_j w_j a_j p_j / B$. For problem instances with $w_j = 1$ for all j, it can be shown that the lower bound is equivalent to a lower bound proposed by Uzsoy (1994) (cf. Webster 1992).

We illustrate the bounds through two example problem instances, one with unit job weights and the other with varying job weights. The first example instance is from Uzsoy (1994).

Example 1. $B = 10$,

j	1	2	3	4	5
p_j	29	46	57	75	95
w_j	1	1	1	1	1
a_j	2	7	4	7	5

Note that $m = 2$ because $\sum_{k=1}^{2} a_{\iota_a(k)} = 2 + 4 \le 10 < 2 + 4 + 5 = \sum_{k=1}^{3} a_{\iota_a(k)}$. Therefore, $C_1 = 29$, $C_2 = 46$, $C_3 = 86$, $C_4 = 121$, $C_5 = 181$, and

$$\begin{aligned} LB_{AK} &= \sum_{j=1}^{n} \left[w_{\iota_w(n-j+1)} \left(C_j - p_j \right) + w_j p_j \right] \\ &= \left[(29 - 29) + (46 - 46) + (86 - 57) + (121 - 75) + (181 - 95) \right] \\ &\quad + \left[29 + 46 + 57 + 75 + 95 \right] = 463. \end{aligned}$$

The optimal sequence for scheduling jobs of length $a_j p_j$ on a single processor to minimize total weighted completion time is $1 - 3 - 2 - 5 - 4$. Therefore,

$$\begin{aligned} F_1^* &= 58 + 286 + 608 + 1083 + 1608 = 3643, \\ F_n^* &= 29 + 46 + 57 + 75 + 95 = 302, \\ R &= \left[58 + 322 + 228 + 525 + 475 \right] / 10 = 160.8, \end{aligned}$$

and

$$LB_S = F_1^*/B + F_n^*/2 - R/2 = \lceil 3643/10 + 302/2 - 160.8/2 \rceil = \lceil 434.9 \rceil = 435.$$

Example 2. $B = 10$,

j	1	2	3	4	5
p_j	29	46	57	75	95
w_j	5	2	1	3	7
a_j	2	7	4	7	5

$$LB_{AK} = \sum_{j=1}^{n} [w_{\iota_w(n-j+1)}(C_j - p_j) + w_j p_j]$$
$$= [7(29-29) + 5(46-46) + 3(86-57) + 2(121-75) + 1(181-95)]$$
$$+ [5(29) + 2(46) + 1(57) + 3(75) + 7(95)] = 1449.$$

The optimal sequence for scheduling jobs of length $a_j p_j$ on a single processor to minimize total weighted completion time is $1-5-2-4-3$. Therefore,

$$F_1^* = 5(58) + 7(533) + 2(855) + 3(1380) + 1(1608) = 11479,$$
$$F_n^* = 5(29) + 2(46) + 1(57) + 3(75) + 7(95) = 1184,$$
$$R = [5(58) + 2(322) + 1(228) + 3(525) + 7(475)]/10 = 606.2,$$

and

$$LB_S = F_1^*/B + F_n^*/2 - R/2 = \lceil 11479/10 + 1184/2 - 606.2/2 \rceil = \lceil 1436.8 \rceil = 1437.$$

In addition to illustrating the mechanics, the numerical examples also help expose the basic character of the two lower bounds. LB_{AK} transforms the batch processor into a parallel machine production system with workload capacity at least as great as the batch processor, then introduces a relaxation in the linkage between jobs and weights. LB_S, on the other hand, is based on a relaxation where jobs are split into smaller jobs in such a way that the batch processor may always be packed to capacity while jobs are sequenced as efficiently as possible.

3. Branch and bound algorithm

The preceding properties and bounds are used in two depth-first B&B algorithms described in this section. The two B&B algorithms differ by the method for computing lower bounds. B&B$_1$ is based on LB_{AK} and B&B$_2$ is based on LB_S.

The branching scheme, which is similar to the scheme proposed by Uzsoy (1994) for the unit weight problem, works as follows. B&B$_1$ computes a lower bound for the root node of the B&B tree as LB_{AK}, whereas B&B$_2$ uses LB_S. If the lower bound is equal to the upper bound as described in section 2, then the algorithm terminates with an optimal schedule. Otherwise, n nodes representing the assignment of each of the n jobs to the first batch emanate from the root node of the B&B tree. At each successive level, the node with the smallest lower bound is selected for branching. There are potentially two nodes emanating from the selected node for each unscheduled job; one node represents the addition of the job to a new batch and the other node represents the addition of the job to the current batch. We refer to the insertion of a job into a new batch as a 'type I addition' and the insertion of a job into a current batch as a 'type II addition'.

Once the node is selected, the first step is to generate type I child nodes that do not satisfy any of the following conditions:

(i) $p(B_{r-1})/w(B_{r-1}) > p(B_r)/w(B_r)$ (otherwise Property 1 is violated);

(ii) there exists an unscheduled job j such that $p_j \le p(B_r)$, $B - a(B_r) \ge a_j$ (otherwise Property 2 is violated);
(iii) there exists two scheduled jobs j and k such that $b(k) < b(j) = r$, $p_j \le p(B_{b(k)})$, $p_k \le p(B_r)$, $w_j \ge w_k$, $B - a(B_{b(k)}) + a_k \ge a_j$, and $B - a(B_{b(j)}) + a_j \ge ak$ (otherwise Property 3 is violated);
(iv) there exists a scheduled job j in batch r and an unscheduled job k such that $b(j) = r$, $p_j < p_k \le p(B_r)$, $w_k \ge w_j$, $a_k \ge a_j$, and $B - a(B_r) + a_j \ge a_k$ (otherwise Property 3 is violated).

Next, in order to maintain feasibility and avoid duplication of batches, we generate only those type II child nodes that do not satisfy either of the following two conditions:

(v) the addition of the unscheduled job violates batch capacity;
(vi) the index of the unscheduled job is smaller than at least one other job in the batch.

If no type I or type II nodes are generated, then the node is fathomed. Otherwise, each new node, with $J =$ the set of unscheduled jobs, requires a lower bound that will be compared to UB to determine if the new node can be fathomed. Both B&B algorithms compute the following lower bound:

$$LB = \sum_{j \notin J} w_j C_j(S) + \sum_{j \in J} w_j \left\{ \sum_{i=1}^{r-1} p(B_i) + \max\{p(B_r), p_j\} \right\}. \tag{1}$$

This lower bound can be computed very quickly. If $LB \ge UB$, then the node is fathomed.

If $LB < UB$, then B&B$_1$ takes an extra step of calculating a second lower bound that is generally tighter. We generalize the interpretation of $\iota_\cdot(k)$ in section 2 to denote a function that returns the index of the 'unscheduled' job with the kth smallest value of parameter $\cdot$. Let x satisfy $\sum_{k=1}^{x} a_{\iota_a(k)} \le B - a(B_r) < \sum_{k=1}^{x+1} a_{\iota_a(k)}$ (i.e. no more than x unscheduled jobs will fit into batch r with the remainder in subsequent batches). B&B$_1$ computes the following lower bound that is based on LB_{AK}:

$$LB = \sum_{j \notin J} w_j C_j(S) + \sum_{j \in J} w_j \sum_{i=1}^{r} p(B_i) + \sum_{j=1}^{|J|-x} w_{\iota_w(|J|-x-j+1)} C_j, \tag{2}$$

where

$$C_j = \begin{cases} p_{\iota_p(j)}, & j \le m \\ C_{j-m} + p_{\iota_p(j)}, & j > m \end{cases}$$

and m satisfies

$$\sum_{k=1}^{m} a_{\iota_a(k)} \le B < \sum_{k=1}^{m+1} a_{\iota_a(k)}.$$

If $LB \ge UB$, then the node is fathomed. Otherwise, the lower bound for the node is set to the maximum of equations (1) and (2).

For B&B$_2$, if $LB < UB$ and the node corresponds to a type I addition, then the following lower bound is computed:

$$LB = \sum_{j \notin J} w_j C_j(S) + \sum_{i=1}^{r-1} p(B_i) \sum_{j \in J} w_j + LB_S, \quad (3)$$

where LB_S is calculated for the jobs in set J. If $LB \geq UB$, then the node is fathomed. Otherwise, the lower bound for the node is set to the maximum of equations (1) and (3). We did not take the extra step of computing equation (3) for nodes created by type II additions because, as we discovered in tests, it generally slows down the algorithm. The reason for this can be seen by observing that equation (3) assumes that processor capacity available to the batch starting at time $\sum_{i=1}^{r-1} p(B_i)$ is $B + \sum_{j \in B_r} a_j$. Thus, as $|B_r|$ increases through type II additions, the lower bound becomes relatively less effective. If an unfathomed node corresponds to a type II addition, then B&B$_2$ sets the lower bound for the node as the maximum of equation (1) and the lower bound of the parent node.

The upper bound specified in section 2 is used until level n is reached in the B&B tree. Nodes at level n define complete schedules, and the best upper bound is updated whenever a better schedule is identified. The procedures terminate when there are no unfathomed partial schedules in the B&B tree.

4. Computational experience

We conducted a computational experiment to assess the effectiveness of the B&B algorithms. The design of the experiment parallels the design used in Uzsoy (1994). We generate random problem instances for $n = 10$, 15, 20 and 25, where $n =$ the number of jobs. Job processing times are drawn from a discrete uniform distribution between 1 and 100. We generate two sets of problem instances that correspond to different methods for generating weights. In the first set, called Set 1, all weights are unity; and in the second set, called Set 2, the weights are drawn from a discrete uniform distribution between 1 and 50. The values of $a_1, a_2, \ldots, a_n$, which are the job sizes, are drawn from a discrete uniform distribution between $a_{\min}$ and $a_{\max}$ with $(a_{\min}, a_{\max}) \in \{(1,5), (4,10), (1,10)\}$ and batch capacity is set to 10. Thirty problem instances are generated for each combination of parameter values. We implemented the two variations of the B&B algorithm in Fortran 77 on an IBM PS/6000 under the Unix operating system.

Tables 1 and 2 report results for weight sets 1 and 2, respectively. For each combination of n and $(a_{\min}, a_{\max})$ we report the average and maximum CPU time and the number of nodes generated. In order to gauge the quality of solutions generated by the heuristic used to generate an initial upper bound (see section 2), we report the average and maximum values of the upper bound divided by the optimum. The last two columns report the average and minimum values of the initial lower bound divided by the optimum. An empty entry in the table indicates that the algorithm was unable to return a known optimal solution within 30 min of CPU time.

The results show that B&B$_1$ outperforms B&B$_2$. As expected, the problem instances in Set 1 are easier to solve than instances in Set 2. We find that when the job weights vary, a small change in the solution is more likely to reduce the total weighted completion time, which in turn causes the best solution to be updated frequently, thereby increasing the solution times. The results also show that the efficiency of the algorithms is sensitive to job size distributions, with increasing job size variation associated with reduced efficiency.

			CPU time		No. of nodes		*UB*/Opt		*LB*/Opt	
n	$(a_{\min}, a_{\max})$	Alg	Avg	Max	Avg	Max	Avg	Max	Avg	Min
10	(1, 5)	B&B$_1$	0.004	0.007	517.1	1225	1.53	2.14	0.66	0.56
	(1, 5)	B&B$_2$	0.006	0.011	1357.3	2211			0.66	0.50
	(4, 10)	B&B$_1$	0.004	0.01	552.6	1504	1.09	1.56	0.68	0.56
	(4, 10)	B&B$_2$	0.015	0.02	1881.7	3368			0.77	0.66
	(1, 10)	B&B$_1$	0.004	0.01	651.2	1474	1.37	1.91	0.58	0.46
	(1, 10)	B&B$_2$	0.011	0.02	2091.1	3788			0.73	0.63
15	(1, 5)	B&B$_1$	0.09	0.17	8827.1	17 501	1.67	2.13	0.61	0.56
	(1, 5)	B&B$_2$	0.23	0.38	39 132.3	62 136			0.70	0.63
	(4, 10)	B&B$_1$	0.25	0.54	25 018.6	60 609	1.13	1.32	0.65	0.56
	(4, 10)	B&B$_2$	0.74	1.24	60 870.0	116 712			0.69	0.63
	(1, 10)	B&B$_1$	0.23	0.43	21 961.5	42 198	1.40	1.88	0.50	0.42
	(1, 10)	B&B$_2$	0.54	0.85	60 626.5	109 817			0.66	0.56
20	(1, 5)	B&B$_1$	2.35	5.95	190 249.9	496 128	1.68	2.28	0.57	0.48
	(1, 5)	B&B$_2$	8.06	17.58	1 165 300.3	2 706 156			0.66	0.54
	(4, 10)	B&B$_1$	10.29	20.57	924 776.1	1 934 630	1.09	1.22	0.64	0.57
	(4, 10)	B&B$_2$	35.27	93.16	1 852 463.3	5 461 575			0.69	0.58
	(1, 10)	B&B$_1$	12.07	26.98	999 022.0	2 183 974	1.37	1.75	0.56	0.47
	(1, 10)	B&B$_2$	25.98	60.88	1 990 265.4	4 258 835			0.69	0.52
25	(1, 5)	B&B$_1$	47.19	134.57	3 064 505.3	8 860 475	1.09	1.38	0.54	0.49
	(1, 5)	B&B$_2$	90.37	163.57	9 010 919.0	18 622 219			0.66	0.59
	(4, 10)	B&B$_1$	909.18	1665.3	31 218 236.5	144 414 247	1.17	1.29	0.63	0.55
	(4, 10)	B&B$_2$								
	(1, 10)	B&B$_1$	576.6	1264.4	41777287.6	105679569	1.38	1.62	0.56	0.48
	(1, 10)	B&B$_2$								

Note: CPU time in seconds.

Table 1. Results for weight set 1: $w_j = 1$.

Columns 8 and 9 in the tables show that the heuristic used to generate an initial upper bound returns schedules that may cost significantly more than the optimum; on average, the heuristic cost premium is $\sim 33\%$ for table 1, 46% for table 2, and $\sim 39\%$ overall. The last two columns show that the initial lower bounds are not tight.

It is interesting to note that, while LB_{AK} appears tighter than LB_S with respect to the overall B&B tree (e.g. compare the average and maximum number of nodes for B&B$_1$ and B&B$_2$ in tables 1 and 2), LB_S generally provides a better initial lower bound. This is due to an overstatement of processor capacity in LB_{AK} that becomes more severe as the number of jobs and the variation and job sizes increases (e.g. see the derivation of m in section 2). The effect is evident in column 8 of tables 1 and 2, i.e. LB_{AK}/Opt is generally decreasing in n and $a_{\max}/a_{\min}$.

We also ran a set of experiments to test the effectiveness of $\max\{LB_{AK}, LB_S\}$ as the lower bound for the root node and type I nodes in the B&B tree. We found that using $LB = \max\{LB_{AK}, LB_S\}$ in place of LB_{AK} does not substantively reduce the number of nodes visited and, more to the point, results in higher CPU time. This is not surprising in light of the relative performance between B&B$_1$ and B&B$_2$ evident in tables 1 and 2, and we thus omit the details.

			CPU time		No. of nodes		*UB*/Opt		*LB*/Opt	
n	$(a_{\min}, a_{\max})$	Alg	Avg	Max	Avg	Max	Avg	Max	Avg	Min
10	(1, 5)	B&B$_1$	0.004	0.011	640.5	1687	1.62	2.40	0.64	0.56
	(1, 5)	B&B$_2$	0.005	0.014	1191.8	2412			0.64	0.48
	(4, 10)	B&B$_1$	0.004	0.009	579.9	1263	1.21	1.70	0.59	0.47
	(4, 10)	B&B$_2$	0.011	0.017	1313.4	2750			0.67	0.56
	(1, 10)	B&B$_1$	0.006	0.016	745.6	2105	1.38	1.87	0.55	0.43
	(1, 10)	B&B$_2$	0.011	0.03	1702.9	5674			0.62	0.49
15	(1, 5)	B&B$_1$	0.13	0.36	13 790.0	43 520	1.84	2.41	0.58	0.44
	(1, 5)	B&B$_2$	0.23	0.48	31 579.1	71 165			0.62	0.55
	(4, 10)	B&B$_1$	0.24	0.80	23 520.8	84 730	1.23	1.71	0.53	0.44
	(4, 10)	B&B$_2$	0.47	1.28	32 327.9	115 688			0.69	0.61
	(1, 10)	B&B$_1$	0.24	0.83	23 673.0	89 703	1.51	2.21	0.47	0.35
	(1, 10)	B&B$_2$	0.40	1.21	38 312.6	128 577			0.66	0.55
20	(1, 5)	B&B$_1$	6.24	24.37	501 976.4	1 936 041	1.61	2.16	0.52	0.47
	(1, 5)	B&B$_2$	14.78	34.31	1 595 594.9	3 804 405			0.65	0.55
	(4, 10)	B&B$_1$	12.55	34.13	1 068 622.8	3 064 970	1.15	1.55	0.49	0.41
	(4, 10)	B&B$_2$	23.49	59.33	990 966.7	2 342 788			0.65	0.53
	(1, 10)	B&B$_1$	22.51	53.33	1 797 630.0	4 295 675	1.41	1.66	0.49	0.40
	(1, 10)	B&B$_2$	28.84	57.74	1 759 005.8	4 378 165			0.67	0.56
25	(1, 5)	B&B$_1$	207.30	622.39	13 304 812.5	43 499 725	1.62	2.19	0.48	0.43
	(1, 5)	B&B$_2$								
	(4, 10)	B&B$_1$								
	(4, 10)	B&B$_2$								
	(1, 10)	B&B$_1$								
	(1, 10)	B&B$_2$								

Note: CPU time in seconds.

Table 2. Results for weight set 2: $w_j \sim \mathrm{U}[1, 50]$.

We tested the effects of Properties 1 through 3 on algorithm performance. Table 3 compares the performance of B&B$_1$ with a modified variation of B&B$_1$ called B&B$_1'$. B&B$_1'$ is the same as B&B$_1$, except conditions (i)–(iv) (see section 3) are not checked to determine whether nodes of type I additions can be discarded. The results show that the improvement achieved by the dominance properties significantly outweighs the extra effort spent in testing them. Further tests, where we excluded fathoming conditions associated with each one of the three properties from the B&B in turn, showed that no single property could be ignored without causing an increase in CPU time.

5. Summary

We have proposed a B&B procedure for a batch processor model that is important because of the economic significance of one of its motivating applications and because it incorporates a complicating factor common in scheduling practice. Insights into this model, and other models like it, are useful building blocks for designing algorithms that apply to more complex and realistic scheduling problems. We find that the B&B algorithm returns optimal solutions for problem instances with up to ~ 25 jobs within 30min of CPU time, and that it can be adapted for use as

Set	$(a_{\min}, a_{\max})$	Alg	CPU time Avg	CPU time Max	No. of nodes Avg	No. of nodes Max
1	(1, 5)	$B\&B_1$	0.09	0.17	8827.1	17 501
	(1, 5)	$B\&B_1'$	0.51	2.04	174 436.2	697 398
	(4, 10)	$B\&B_1$	0.25	0.54	25 018.6	60 609
	(4, 10)	$B\&B_1'$	5.69	24.43	2 907 179.4	12 410 235
	(1, 10)	$B\&B_1$	0.23	0.43	21 961.5	42 198
	(1, 10)	$B\&B_1'$	3.79	18.99	1572693.1	8 371 284
2	(1, 5)	$B\&B_1$	0.13	0.36	13 790.0	43 520
	(1, 5)	$B\&B_1'$	1.00	4.95	455 540.7	2 371 077
	(4, 10)	$B\&B_1$	0.24	0.80	23 520.8	84 730
	(4, 10)	$B\&B_1'$	4.44	20.88	2 162 535.2	9 976 354
	(1, 10)	$B\&B_1$	0.24	0.83	23 673.0	89 703
	(1, 10)	$B\&B_1'$	3.22	16.33	1 507 886.5	7 609 899

Note 1. $B\&B_1'$ was not able to return solutions within 30 minutes of CPU time when $n = 20$.
Note 2. CPU time in seconds.

Table 3. Results $n = 15$[1].

a heuristic to save CPU time and/or use as a tool for larger problem instances. Worthwhile extensions of the model for future research include consideration of incompatible job families and due date performance criteria.

References

AZIZOGLU, M. and KIRCA, O., 1999, On the minimization of total flow time with identical and uniform parallel machines. *European Journal of Operational Research*, **113**, 91–100.

BRUCKER, P., GLADKY, A., HOOGEVEEN, H., KOVALYOV, M. Y., POTTS, C. N., TAUTENHAHN, T. and VAN DE VELDE, S. L., 1998, Scheduling a batching machine. *Journal of Scheduling*, **1**, 31–54.

CHANDRU, V., LEE, C.-Y. and UZSOY, R., 1993a, Minimizing total completion time on a batch processing machine with job families. *Operations Research Letters*, **13**, 61–65.

CHANDRU, V., LEE, C.-Y. and UZSOY, R., 1993b, Minimizing total completion time on batch processing machines. *International Journal of Production Research*, **31**, 2097–2121.

DUPONT, L. and GHAZVINI, F. J., 1997, Branch and bound method for single batch processing machine with mean flow time criteria. *International Journal of Industrial Engineering: Practice and Theory*, **4**, 197–203.

HOCHBAUM, D. S. and LANDY, D., 1997a, Scheduling with batching: two job types. *Discrete Applied Mathematics*, **72**, 99–114.

HOCHBAUM, D. S. and LANDY, D., 1997b, Algorithms and heuristics for scheduling semiconductor burn-in operations. *Operations Research*, **45**, 874–885.

IKURA, Y. and GIMPLE, M., 1986, Efficient scheduling algorithms for a single batch processing machine. *Operations Research Letters*, **5**, 61–65.

MEHTA, S. V. and UZSOY, R., 1998, Minimizing total tardiness on a batch processing machine with incompatible job families. *IIE Transactions*, **30**, 165–178.

SCHWIEGELSHOHN, U., LUDWIG, W., WOLF, J. L., TUREK, J. and YU, P. S., 1998, Smart SMART bounds for weighted response time scheduling. *SIAM Journal on Computing*, **28**, 237–253.

UZSOY, R., 1994, Scheduling a single batch processing machine with non-identical job sizes. *International Journal of Production Research*, **32**, 1615–1635.

UZSOY, R., 1995, Scheduling batch processing machines with incompatible job families. *International Journal of Production Research*, **33**, 2685–2708.

UZSOY, R., LEE, C. Y. and MARTIN-VEGA, L. A., 1992, A review of production planning and scheduling models in semiconductor industry–Part I: system characteristics, performance evaluation and production planning. *IIE Transactions*, **24**, 47–61.

UZSOY, R. and YANG, Y., 1997, Minimizing total weighted completion time on a single batch processing machine. *Production and Operations Management*, **6**, 57–73.

WEBSTER, S., 1992, New bounds for the identical parallel processor weighted flow time problem. *Management Science*, **38**, 124–136.

WEBSTER, S., 2000, Frameworks for adaptable scheduling algorithms. *Journal of Scheduling,* **3**, 21–49.